Measuring Engineering Report

Hannah Mahon

17329071

# Table of Contents:

Introduction……………………………………………………………………………………………………………………2

Measurable Data…………………………………………………………………………………………………………….3

Computational Platforms………………………………………………………………………………………………..5

Algorithmic Approaches………………………………………………………………………………………………….9

Ethics…………………………………………………………………………………………………………………………....11

Conclusion…………………………………………………………………………………………………………………….12

Bibliography.…………………………………………………………………………………………………………………13

# Introduction:

*Software Engineering refers to the disciplined application of engineering, scientific, mathematical methods used to produce good quality software design (Humphrey, 1988).*

The above quote sums up the reasoning behind software engineering. However, these intricate processes are hard to measure. It is difficult to place figures on the quality and development of a project. Nonetheless, there are a few tried and tested metrics that are in place to analyse different parts of a development projects such as the number of lines of code written or the number of commits to a project. This report aims to first highlight some of the popular metrics used in modern day software design. Following from that, the computational platforms used to analyse these metrics for organisations will be compared and discussed. Next, the report will delve into the different algorithms that are accessible for calculating these different metrics. Finally, the last part of the report will deliberate the ethics behind measuring engineering as a process, in a world where GDPR has taken precedence, is taking employees data to calculate these metrics too controversial?

The world of business is becoming much more competitive and markets are becoming more and more saturated. By measuring engineering and defining what productivity in the workplace is, organisations can become more efficient and productive, giving them an edge over their competitors. Measuring engineering is not only important but now is vital to organisational growth and development.

# Measurable Data:

Productivity is usually defined as “a ratio of outputs units produced to the input units of effort” (Sudhakar et al, 2012). Productivity is important to measure as it can cut down the overall costs of software development and can also highlight the top performers within a firm (Sudhakar et al, 2012).

Productivity in software engineering is traditionally measured by the following metrics; Lines of code (LOC) or Source lines of code (SLOC), the number and quality of commits, personal software processes, lead time, code churn and JUnit testing to name a few.

## Lines of Code LOC/SLOC:

Lines of code or source lines of code is a software metric to predict the amount of time it will take to develop a program by counting the lines of code written (Park, 1992). It is one of the oldest software metrics however its importance is still present nowadays. It is useful as it is a measure of development/maintenance effort, as well as a variable for other software engineering and finally as a standard against other metrics (Rosenberg, n.d.)

## The number and quality of Commits:

Although there is no definition of what a small or large commit is, there is a rough guideline suggested that small commits are classified as those with one-line changes and large commits being the 99th percentile of all commits (Hattori and Lanza, 2008). Some researchers look favourably on small commits as they are more likely to be well-defined tasks and it is easier to understand the intentions behind the commit itself. However large commits are useful in giving an insight into the way in which projects are being developed and show the software development skills of its developer (Hindle, German and Holt, 2008).

## Personal Software Processes:

The personal software process is a collection of techniques that help software engineers improve their performance and provides tools for their employers to measure and analyse their work. Research has shown there are many benefits to the personal software processes (PSP) such as; fewer code bugs, better estimating and planning and enhanced productivity. PSP is a seven-step program which has four levels including; personal measurement, personal planning, personal quality and cyclic process. Personal measurement is the first part of the process where engineers learn how to measure development, time and defects. In the personal planning phase, it introduces the PROBE method. This is used by developers to estimate the size and development time for new programs based on previous data. Personal quality introduces defect management where engineers develop and use checklists for design and review. This is to ensure that they can effectively review and modify their own code. Lastly the cyclic process illustrates to developers how the whole process can be made into a cycle which can be used from project to project (Humphrey, 1996).

## Lead time:

Lead time is an agile software measure that calculates the time taken from the start of the work through to the end of the process and takes into account waiting time, set up time, real operating time and post-processing time. This metric is favourable in conditions where the code needs to be developed quickly. It is used to estimate the time that the project is going to take on average. Clients would want this metric calculated as they would benefit from knowing when the work is estimated to be finished (Stackify, 2019).

## Code Churn:

Code Churn is how often a file has evolved over a certain period. It is used to show how what parts of the code needs more testing because they’re being changed more often and the parts of the code that are getting more development resources. Ideally, you want a lower code churn as its closer to a release date of being finished. High code churn means that a lot of your code needs testing or reviewing. High code churn can be caused by continually changing requirements, exploratory coding; if code is just being developed to learn a new skill and lastly indecisive developers; if the developers on a team have different strategies for the project and can’t come to a conclusive plan (TextExpander.com, 2019).

## Junit Testing:

Software testing is a metric that is widely used by developers. It is a piece of software written that executed another piece of software and validates the code by seeing if it passes those tests. This helps the developer to validate their code from the testing. Another metric is built into the Junit testing which tests code coverage. This can highlight the inefficiencies within the code by showing the developer which lines of code aren’t being reached (c, 2007, 2019).

Testing code can increase programming speed and improve the overall quality of the code. This contributes to more efficient and productive development teams which is why unit testing is a favourable metric within organisations (Tutorialspoint.com, 2019).

Although these metrics are in place to measure productivity, there are a number of different factors that can influence productivity within the development of software that metrics cannot account for. These include; the capability of the programmer, team cohesion, platform experience, which languages the programmer is proficient in, tools and software experience and analyst capability. Setting goals to the programmer, providing training and giving constructive feedback have all been proven to improve overall productivity.

The software firm itself can also influence productivity positively by having sufficient computer infrastructure, useful software engineering tools especially for testing, document management systems, object orientated programming techniques and well-organised teams to name a few (Sudhakar et al, 2012)

# Computational Platforms:

The number of computational platforms has grown massively in recent years as the software industry has become more competitive and saturated. The insights provided by these platforms can give organisations a competitive edge and enhance their productivity massively. They also highlight the problems that have occurred with the code, which employees are over-worked or not working enough, and they give information about the team’s efficiency as a whole (Rezvina, 2019).

A few popular platforms include GitPrime, Wrike and Gitential.

## GitPrime:

GitPrime is an analytic tool that analyses data from different codebases to produce comprehensive reports about the progress of a software development project. It makes use of indexes, metrics and repositories from Git-based codebases such as Github, GitLab and Bitbucket. It is convenient for managers as they can view the project as a whole from employees to code in real-time and make adjustments to improve as they go along (FinancesOnline.com, 2019).

The platform includes pull request charts where you can visualise the comments and volume of pull requests. These also help to manage pull requests at scale and review the team’s dynamics. GitPrime also has resolution insights that allows you to see trends overtime within the team which can be used to set meaningful targets.

The metrics for team leaders include; responsiveness: how often are developers responding to comments, comments addressed: what is the frequency of comments responders, receptiveness: how often is a developer submitting and unreviewed pull requests: are developers merging code without reviewing.

The metrics for developers include; time to first comment: are they responding to new pull requests, influence: are comments leading to follow-up commits; review coverage: are reviewers commenting on edits and reaction time: how long does it take to address new comments (Gitprime.com, 2019).

A screenshot of a cell phone

Description automatically generated

Figure 1:

An example of the pull request visualisation

A screenshot of a social media post

Description automatically generated

Figure 2:

Insights from a developer’s point of view

A screenshot of a cell phone

Description automatically generated

Figure 3:

Insights from a team leaders’ point of view

## Wrike:

Wrike is software built for online project management that claims to improve the productivity and efficiency of work of project teams (Financesonline.com, 2019). It helps users with projects from start to finish providing full visibility and control over the project. The team leader can visualise the project requirements, plans, schedules and the insights on a Gantt chart. Each team member can clearly see what their priorities and tasks are. It differs from GitPrime slightly as it also as a mobile app and is typically aimed at bigger businesses. Many Fortune 500 companies such as Google and Adobe are advocates for this platform (Inc. et al.,2019).

The features of Wrike include: dynamic request forms which ensures teams have all project details before starting, a personal dashboard for the user, a live editor, document version control, time and budget tracking, Gantt charts which can identify conflicts, project and team reporting and IOS/Android apps. It also provides insights into the task management giving both developers and team leaders shareable reports which improve and enhance productivity for both parties (Wrike, 2019).

It is also possible to sync Wrike with Github so you can collaborate on across both platforms. Comments addressed in Wrike will automatically upload to Github and vice versa (Wrike, 2019).

A screenshot of a cell phone

Description automatically generated

Figure 4:

The task management insights

A screenshot of a social media post

Description automatically generated

Figure 5: Github and Wrike integrated.

## Gitential

Gitential is a software tool designed to “measure and visualise the productivity and efficiency of your software development based on Git activity” (Gitential.com, 2019). The software works with git repositories only and can also help to identify possible issues within the code.

Gitential doesn’t measure the lines or quality of the code, it works by measuring the team’s coding hours, coding volume and utilisation of the code. This allows the team to enhance their productivity and see what they can improve on (JAXenter, 2019).

Researches at Gitential have shown that transparency when it comes to metrics can have a positive effect on performance and has shown to improve productivity and performance substantially (Blog.gitential.com, 2019).

# A screenshot of a computer screen Description automatically generated

Figure 6: insights from Gitential from a team leaders’ point of view. Can see that the developer Oma’s efficiency is below team average and has disimproved.

A screenshot of a computer

Description automatically generated

Figure 7: Gitential displays the teams coding hours by developer. Team leaders can easily see which developers are the most productive in terms of hours worked.

# Algorithmic Approaches:

The use of algorithms such as link algorithms, Cyclomatic Complexity and Halstead’s software science all contribute to measure performance. Depending on the metric you want to analyse, these three algorithms for example can be used to calculate them. However, these algorithms alone cannot solely be used as performance is much more than just numbers. As said previously, the organisation itself has a massive influence on performance that algorithmic approaches alone cannot decipher.

## Link Algorithms:

Link algorithms have been used to show the amount of pull requests a developer has access to and the amount of commits they have uploaded (Hackernoon.com, 2019). Link algorithms work by evaluating the connections between nodes. There are many types of link algorithms such as Dijsktra’s algorithm, breadth first search and depth first search. In measuring productivity, the different chains within these algorithms express the different repositories within the organisation and the number of commits by each repository (Hackernoon.com, 2019).

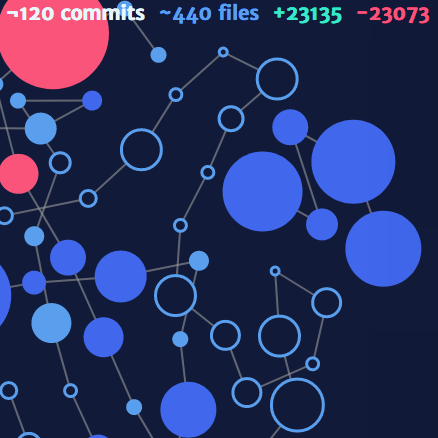


Figure 8: Visualisation of a link algorithm showing the different repositories within an organisation

Cyclomatic Complexity

Cyclomatic complexity is a measure of the maximum number of linearly independent paths in a program (Gill and Kemerer, 1991). In simple terms, it counts the number of decisions in the program, the higher the count, the more complex the program. It is used to simplify the code and to decipher the tests required. The following formula is used to calculate cyclomatic complexity: CYC = E – N + 2P where E = number of edges, N = number of nodes and P = number of disconnected parts of the flow diagram (subroutines). If and while statements for example would be considered +1 to complexity (Perforce Software, 2019).

Lower complexity programs are considered good practice as the higher the complexity the more issues that are likely to arise and it is also more challenging to test (Perforce Software, 2019).

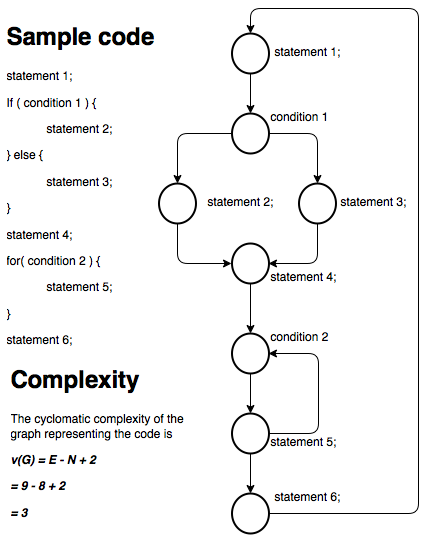


Figure 9: Example of how the cyclomatic complexity algorithm works.

## Halstead’s Software Science:

Software science measures the complexity of programs by the number of references to operators and operands. There are 4 fundamental measurements: n1 = number of unique operators, n2 = number of unique operands, N1 = total number of operator occurrences, N2 = total number of operand occurrences (Hamer and Frewin, 1982).

Using software science, you can use the number of operators and operands to measure metrics such as program length, the program vocabulary and the program volume. The program length is calculated by N1 + N2, the program vocabulary is calculated by n1+n2, and the program volume is calculated by Nlog2n. Program difficulty and effort can also be calculated using the operators and operands. Program difficulty is measured by n1/2 \* N2/n2 and effort can be measured by D \* V (difficulty \* volume).

This metric although useful has its flaws. The metric cannot account for large multi-module programs and fails to consider that project duration is more than just coding (Hamer and Frewin, 1982).

# Ethics:

Measuring engineering is a newfound phenomenon that has changed organisations efficiency and performance positively. However, there is some ethical issues surrounding the metrics used.

The first ethical issue that can arise is the negative connotations surrounding performance appraisal. If employees get a negative review from their boss as a result of a bad result on one of the metrics used, this could hinder their performance going forward. It could impact massively on their self-esteem, performance and motivation thereafter. Employees tend to blame external factors such as the organisation before putting the blame on themselves (Pearce and Porter, 1986). Focusing on just the numbers produced by these metrics, organisations can lose sight on employee wellbeing impacting massively on the organisation as a whole. After all, employees are the fundamental resource of the organisation (Drucker, 2002).

Another major ethical issue that is associated with measuring performance is the exploitation of employee’s data for the calculations of these metrics. Data collection is a very controversial topic nowadays, with the introduction of the GDPR rules. Organisations must make sure they are compliant with these rules while calculating metrics. Employees might be weary of their employers collecting data on them. This might spark distrust between employees and employers. Employers must be completely transparent with the data they’re retrieving. Access to this data should be limited, with only top executives being allowed to see.

Personally, I think that measuring engineering is important to an extent. It is important to see how developers are working together and taking part in the project. Measuring the efficiency of the team has been proven to promote productivity within the workforce. However, it is vital to not become obsessed with the numbers that the metrics produce. Developers are people with potential regardless of the number of lines of code they produce.

# Conclusion:

This report identified many ways of measuring productivity within software development such as number of lines of code produced, number of commits, personal software processes, lead time, code churn and JUnit testing. However, these metrics alone cannot be solely used to calculate productivity as a whole. Organisations can influence productivity in many more ways that cannot be calculated such as using teams and having sufficient computer infrastructure.

The computational platforms such as Gitprime, Wrike and Gitential were compared and discussed. These platforms give executives an overview of the development project as a whole and how each developer on the team is getting on with the project itself. Each one of these platforms have similar design features but show different metrics on the developers. These platforms are frequently deployed by organisations as they have been proven to boost efficiency and productivity within the firm.

The algorithms analysed were link algorithms such as Dijkstra to measure interactions between repositories, cyclomatic complexity to count the number of independent paths in a program to define how complex a program is and lastly the Halstead software science algorithm to measure the complexity of programs as well. These algorithmic approaches have been used throughout software engineering and are not a new phenomenon. They can be efficient in measuring engineering however like most algorithms, they are just an estimate.

Lastly, the report delved into the ethical stance behind measuring engineering. As discussed, in the world of GDPR, companies must be completely transparent with the data they are collecting. They also must remember the metrics are only estimates, there are many unmeasurable ways of being productive within a software project.

Measuring software engineering is a complex task, with different researchers promoting a different metric. Although these metrics are crucial for organisations to observe and analyse, they must remember that they are estimates. Their employees are more than just a number produced by a metric. If organisations solely focus on these, it can have the opposite effect than intended and hinder performance by decreasing motivation and self-esteem. However, when metrics are used with other performance standards, they have great potential for an organisation to flourish (Sudhakar et al., 2012).

# Bibliography:

Sudhakar, P., Farooq, A. and Patnaik, S. (2012). Measuring productivity of software development teams. *Serbian Journal of Management*, 7(1), pp.65-75.

Gousios, G., Kalliamvakou, E. and Spinellis, D. (2008). Measuring Developer Contribution from Software Repository Data. *ACM Digital Journal*.

Rosenberg, J. (n.d.). Some misconceptions about lines of code. *Proceedings Fourth International Software Metrics Symposium*.

Park, R. (1992). Software Size Measurement: A Framework for Counting Source Statements. *Software Engineering Institute*.

Hattori, L. and Lanza, M. (2008). On the nature of commits. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*.

Hindle, A., German, D. and Holt, R. (2008). What do large commits tell us?. *Proceedings of the 2008 international workshop on Mining software repositories - MSR '08*.

Humphrey, W. (1996). Using a defined and measured Personal Software Process. *IEEE Software*, 13(3), pp.77-88.

Rezvina, S. (2019). *Velocity vs. GitPrime: Choosing an Engineering Intelligence Tool*. [online] Code Climate. Available at: https://codeclimate.com/blog/velocity-vs-gitprime/ [Accessed 31 Oct. 2019].

Financesonline.com. (2019). *GitPrime Reviews: Pricing & Software Features 2019 - Financesonline.com*. [online] Available at: https://reviews.financesonline.com/p/gitprime/ [Accessed 31 Oct. 2019].

Gitprime.com. (2019). [online] Available at: https://www.gitprime.com/platform/pull-request/ [Accessed 31 Oct. 2019].

Financesonline.com. (2019). *Wrike Reviews: Pricing and Software Features of wrike.com | FinancesOnline*. [online] Available at: https://reviews.financesonline.com/p/wrike/ [Accessed 31 Oct. 2019].

Inc., W., George, P., Krein, M. and Dance, J. (2019). *Wrike Pricing, Features, Reviews & Comparison of Alternatives*. [online] GetApp. Available at: https://www.getapp.com/project-management-planning-software/a/wrike/ [Accessed 31 Oct. 2019].

Wrike. (2019). *Getting started with project management tools – Wrike Tour*. [online] Available at: https://www.wrike.com/tour/ [Accessed 31 Oct. 2019].

Gitential.com. (2019). *Analytics for Git*. [online] Available at: https://gitential.com/ [Accessed 2 Nov. 2019].

JAXenter. (2019). *Gitential: Bringing analytics to software development - JAXenter*. [online] Available at: https://jaxenter.com/gitential-140924.html [Accessed 2 Nov. 2019].

Blog.gitential.com. (2019). *Blog | Gitential.com*. [online] Available at: https://blog.gitential.com/ [Accessed 2 Nov. 2019].

Hackernoon.com. (2019). *Measuring Developer Productivity*. [online] Available at: https://hackernoon.com/measure-a-developers-impact-e2e18593ac79 [Accessed 2 Nov. 2019].

Gill, G. and Kemerer, C. (1991). Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*, 17(12), pp.1284-1288.

Perforce Software. (2019). *What Is Cyclomatic Complexity? | Perforce Software*. [online] Available at: https://www.perforce.com/blog/qac/what-cyclomatic-complexity [Accessed 2 Nov. 2019].

Hamer, P. and Frewin, G. (1982). M.H. Halstead's Software Science - a critical examination. *ICSE '82 Proceedings of the 6th international conference on Software engineering*, pp.197-206.

Pearce, J. and Porter, L. (1986). Employee responses to formal performance appraisal feedback. Journal of Applied Psychology, 71(2), pp.211-218.

Drucker, P. (2002). They're not Employees, They're People. Harvard Business Review, 80(2).

Humphrey, W. (1988). The software engineering process: definition and scope. *Proceedings of the 4th international software process workshop on Representing and enacting the software process -*.

TextExpander. (2019). *What is Code Churn and How to Reduce it - TextExpander*. [online] Available at: https://textexpander.com/entry/what-is-code-churn-and-how-to-reduce-it/ [Accessed 6 Nov. 2019].

(c) 2007, 2. (2019). *Unit Testing with JUnit - Tutorial*. [online] Vogella.com. Available at: https://www.vogella.com/tutorials/JUnit/article.html [Accessed 6 Nov. 2019].

Tutorialspoint.com. (2019). *JUnit - Test Framework - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/junit/junit\_test\_framework.htm [Accessed 6 Nov. 2019].

Stackify. (2019). *15 Metrics for DevOps Success*. [online] Available at: https://stackify.com/15-metrics-for-devops-success/#post-14669-\_jx3advhf4n5g [Accessed 6 Nov. 2019].